Robot Open Standard - example

version 6.0

Table of Contents

INTRODUCTION

The manual is intended for programmers who want to take advantage of the *Robot Millennium* calculation kernel in their applications. It assumes that the reader has basic knowledge of programming in Visual Basic v. 6.0 with the use of *COM* technology. The manual is arranged in the form of comprehensive commentary to the code of the example application, which carries out calculations of simple frame structure with slabs.
To run the example the User has to have V*isual Basic* or *Visual C++* version 6.0 or more recent, and installed *Robot Millennium* version 17.5.1.1764 or more recent.

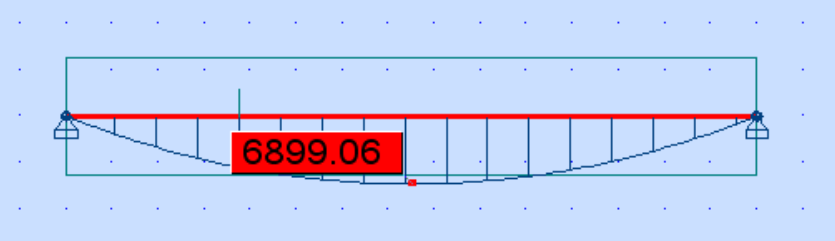
Description of COM technology

Robot Open Standard (ROS) is provided in the form of COM component. In order to be able to use the component, the User should add *RobotOM.tlb* library to his application. If the application is developed in *Visual Basic*, it is done by adding the reference to *Robot Open Standard* component by means of command: *Menu/Project/References*.
In case of applications developed in C++, the library should be added directly in the application code by means of the following command:

```
#import <RobotOM.tlb>
```

The simplest application

The example below carries out calculations of one-span beam with pinned supports on both ends and displays the span moment due to the self-weight load.
Attention! The example below, as well as the further ones, will work only with legal version of *Robot Millennium* with the appropriate protection dongle.




```
Dim Robot As New RobotApplication
Robot.Project.New I_PT_FRAME_2D

Robot.Project.Structure.Nodes.Create 1, 0, 0, 0
Robot.Project.Structure.Nodes.Create 2, 3, 0, 0

Robot.Project.Structure.Bars.Create 1, 1, 2

Dim Label As RobotLabel
Set Label = Robot.Project.Structure.labels.Create(I_LT_SUPPORT,
"Support")
Dim SupportData As RobotNodeSupportData
Set SupportData = Label.Data
SupportData.UX = 1
SupportData.UY = 1
SupportData.UZ = 1
```

```
SupportData.RX = 0
SupportData.RY = 0
SupportData.RZ = 0
Robot.Project.Structure.labels.Store Label
Robot.Project.Structure.Nodes.Get(1).SetLabel I_LT_SUPPORT, "Support"
Robot.Project.Structure.Nodes.Get(2).SetLabel I_LT_SUPPORT, "Support"

Set Label = Robot.Project.Structure.labels.Create(I_LT_BAR_SECTION,
"Beam 50*50")
Dim section As RobotBarSectionData
Set section = Label.Data
section.ShapeType = I_BSST_CONCR_BEAM_RECT
Dim concrete As RobotBarSectionConcreteData
Set concrete = section.concrete
concrete.SetValue I_BSCDV_BEAM_B, 0.5
concrete.SetValue I_BSCDV_BEAM_H, 0.5
section.CalcNonstdGeometry
Robot.Project.Structure.labels.Store Label
Robot.Project.Structure.Bars.Get(1).SetLabel I_LT_BAR_SECTION, "Beam
50*50"

Dim caseSW As RobotSimpleCase
Set caseSW = Robot.Project.Structure.Cases.CreateSimple(1, "SW",
I_CN_PERMANENT, I_CAT_STATIC_LINEAR)
caseSW.Records.New I_LRT_DEAD
Dim LoadRec As RobotLoadRecord
Set LoadRec = caseSW.Records.Get(1)
LoadRec.SetValue I_DRV_Z, -1
LoadRec.SetValue I_DRV_ENTIRE_STRUCTURE, True

If Robot.Project.CalcEngine.Calculate = True Then
    MsgBox Robot.Project.Structure.Results.Bars.Forces.Value(1, 1,
0.5).MY
End If
```

STRUCTURE MODELING

We start the work with the structure in *ROS* by creating the new project and setting the preferences. The manual cannot cover the detailed description of all options. In the example it is also presented how to parametrize the task, this method may successfully be applied in calculations of real structures, consisting of such elements as: beams, columns, slabs, walls, continuous footing, spread footing (also on elastic ground), loaded by different types of dead and live loads, wind or seismic loads.

The application code begins with creating *ROS* object model, applying the structure type – in this case it is 3D structure with 6 degrees of freedom.

```
Dim Robot As New RobotApplication
Robot.Project.New I_PT_SHELL
```

Setting the project preferences

Project preferences include many options such as bill of material of which the elements are made, method of mesh generation, the set of national standards, according to which the calculations are performed, etc. Most of them may be remained unchanged, using the default settings. The example below selects ”BAEL 91” national standard for calculations of slab reinforcement.

```
Dim ProjectPrefs As RobotProjectPreferences
Set ProjectPrefs = Robot.Project.Preferences
ProjectPrefs.SetActiveCode I_CT_RC_THEORETICAL_REINF, "BAEL 91"
```

Parameters of mesh generation

One of the most important options, having the significant influence on structure analysis, is method of mesh generation in FEM analysis. The options set in the example below work for most of structures. The average size of finite element is set as 0,5 m, what ensures the sufficient precision and satisfactory speed of calculations. Reducing the size of finite element considerably lengthens time of calculations and therefore the optimal size should be set empirically for each structure.

```
Dim MeshParams As RobotMeshParams
Set MeshParams = ProjectPrefs.MeshParams
MeshParams.SurfaceParams.Method.Method = I_MMT_DELAUNAY
MeshParams.SurfaceParams.Generation.Type = I_MGT_ELEMENT_SIZE
MeshParams.SurfaceParams.Generation.ElementSize = 0.5
MeshParams.SurfaceParams.Delaunay.Type = I_MDT_DELAUNAY
```

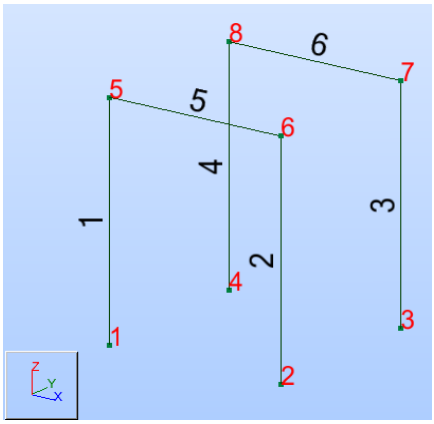
Modeling the members (beams, columns)

We create the linear elements, such as beams and columns defining their axis and section, the element axis is located by default in the center of gravity of section. The element axes are spread between the nodes and therefore we start their definition from defining the nodes, and next we define the members, giving their begin and end in the form of node numbers. Each node and member has its number, starting from 1; this number should be given in course of definition. The numbering of nodes and members is independent.

To facilitate the work we may use the function returning the successive available number for the given type of object, in order to prevent the possible collision of numbering. These numbers should however be remembered, in order to enable subsequent getting of results of these elements.

In the example below 8 nodes and 6 members spreading between these nodes (4 columns and 2 beams) were defined. We define the nodes giving the successive available number and 3 coordinates *X, Y, Z* to each them, and the members giving the successive available number and the numbers of nodes representing their ends. The structure model with numbers is presented on the drawing below.

```
With str.Nodes
    .Create 1, 0, 0, 0
    .Create 2, 3, 0, 0
    .Create 3, 3, 3, 0
    .Create 4, 0, 3, 0
    .Create 5, 0, 0, 4
    .Create 6, 3, 0, 4
    .Create 7, 3, 3, 4
    .Create 8, 0, 3, 4
End With

With str.Bars
    .Create 1, 1, 5
    .Create 2, 2, 6
    .Create 3, 3, 7
    .Create 4, 4, 8
    .Create 5, 5, 6
    .Create 6, 7, 8
End With
```



Section definition

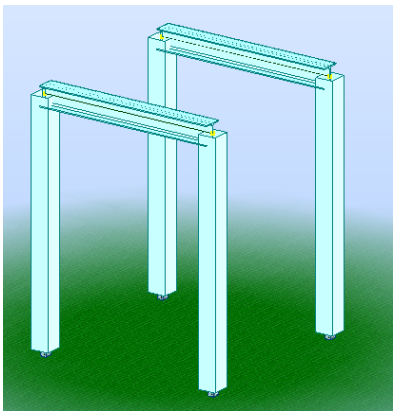
The User should assign the sections to the defined members. One section may be assigned to the optional number of members. The most often used sections are steel and concrete sections, which are defined in a different way because steel sections are selected from the base of steel sections whereas the concrete sections should be defined by giving their size every time.

In the example below we define one square concrete section of 30cm of size and one steel section named ”HEA 340” basing on ”RCAT” section database.

The concrete section is then assigned to the columns (members numbered 1 to 4), and steel section is assigned to the beams (members numbered 5 and 6). Materials, from which the elements of the given sections are made, will get default parameters selected by the program. It is possible to change these parameters (description provided in the next chapter).

```
Dim labels As RobotLabelServer
Set labels = str.labels

ColumnSectionName = "Rect. Column 30*30"
Set Label = labels.Create(I LT BAR SECTION, ColumnSectionName)
Dim section As RobotBarSectionData
Set section = Label.Data
section.ShapeType = I_BSST_CONCR_COL_R
Dim concrete As RobotBarSectionConcreteData
Set concrete = section.concrete
concrete.SetValue I BSCDV COL B, 0.3
concrete.SetValue I_BSCDV_COL_H, 0.3
section.CalcNonstdGeometry
labels.Store Label

Dim selectionBars As RobotSelection
Set selectionBars = str.Selections.Get(I_OT_BAR)
selectionBars.FromText ("1 2 3 4")
str.Bars.SetLabel selectionBars, I_LT_BAR_SECTION, ColumnSectionName

Dim steelSections As RobotSectionDatabaseList
Set steelSections = ProjectPrefs.SectionsActive
If steelSections.Add("RCAT") = False Then
    Output.AddItem "Steel section base RCAT not found..."
End If

selectionBars.FromText ("5 6")
Set Label = str.labels.Create(I LT BAR SECTION, "HEA 340")
str.labels.Store Label
```

```
str.Bars.SetLabel selectionBars, I_LT_BAR_SECTION, "HEA 340"
```



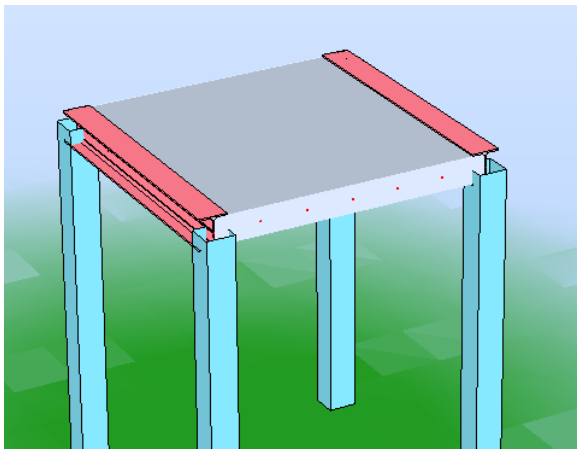
Material definition

In the real projects it is necessary to define the materials such as concrete, steel, timber with their resistance parameters. The example below presents the method of material definition (concrete type). This material may be later applied to any structural element, having the concrete section. In the next chapter this material will be applied to the defined slab.

```
MaterialName = "Concrete 30"

Set Label = labels.Create(I_LT_MATERIAL, MaterialName)

Dim Material As RobotMaterialData
Set Material = Label.Data
Material.Type = I MT CONCRETE
Material.E = 30000000000# ' Young
Material.NU = 1# / 6# ' Poisson
Material.RO = 25000# ' Unit weight
Material.Kirchoff = Material.E / (2 * (1 + Material.NU))
labels.Store Label
```

Modeling the panels (of slabs, walls)

We define the slabs by giving the list of points with coordinates, which create the slab contour. As opposed to the members, the contour does not spread on the existing structural nodes, but it is necessary to give the node coordinates directly and basing on them the program will automatically generate the structural nodes. The slab contour should be closed, i.e. the first point of the contour should be equal to the last one. In case of rectangular slab, the contour should contain 5 points creating the contour corners and the first point should be equal to the fifth one. It is very important that all contour points are in the same plane. Except for contour definition, we should assign thickness (section) and material to each slab. The contour plane defines the middle of the slab section.



Preparing the list of points defining the slab contour; in the example below the contour coordinates are equal to the coordinates of previously defined nodes, program will automatically connect the slab contour with appropriate nodes in course of structural mesh generation.

```
Dim points As RobotPointsArray
Set points = Kernel.CmpntFactory.Create(I CT POINTS ARRAY)
points.SetSize 5
With points
    .Set 1, 0, 0, 4
    .Set 2, 3, 0, 4
    .Set 3, 3, 3, 4
    .Set 4, 0, 3, 4
    .Set 5, 0, 0, 4
End With
```

Defining the slab section (thickness). This section may be assigned to any number of slabs.

```
SlabSectionName = "Slab 30"
Set Label = labels.Create(I LT PANEL THICKNESS, SlabSectionName)
Dim thickness As RobotThicknessData
Set thickness = Label.Data
thickness.MaterialName = MaterialName
thickness.ThicknessType = I_TT_HOMOGENEOUS
Dim thicknessData As RobotThicknessHomoData
Set thicknessData = thickness.Data
thicknessData.ThickConst = 0.3
labels.Store Label
```

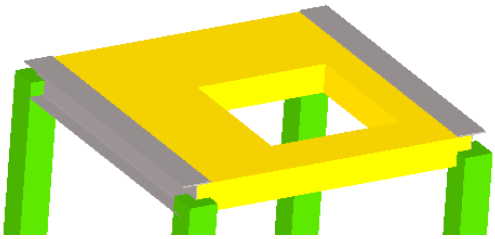
Defining the slab object and assigning the attributes: geometry and section to it. Additionally we should set *Meshed* parameter to *True*, what means that the slab is the integral part of the structure and will be meshed. Non-meshed contours are used for example in the hole definition or as the auxiliary objects in the contour load definition.

Defining the contour we should give its number. Numbering of contours and members is common and therefore we should make sure that the numbers of members and slabs do not collide with one another. We may take advantage of *FreeNumber* method, returning the successive available object number, which may be used in course of object definition.

```
Dim slab As RobotObjObject
objNumber = str.Objects.FreeNumber
str.Objects.CreateContour objNumber, points
Set slab = str.Objects.Get(objNumber)
slab.Main.Attribs.Meshed = True
slab.SetLabel I LT PANEL THICKNESS, SlabSectionName
slab.Initialize
```

Hole definition

The hole definition in the slabs is similar to the slab definition, however we should remember about a few principles, which help to avoid the possible errors that may occur in course of calculations. The hole contour should be located inside the slab contour and in its plane. We may also define the holes comprising the several contacting slabs.
We should assign *Meshed* = *False* attribute to the holes without assigning the section.



```
With points
    .Set 1, 1.1, 1.1, 4
    .Set 2, 2.5, 1.1, 4
    .Set 3, 2.5, 2.5, 4
    .Set 4, 1.1, 2.5, 4
    .Set 5, 1.1, 1.1, 4
End With

Dim Hole As RobotObjObject
HoleNumber = str.Objects.FreeNumber
str.Objects.CreateContour HoleNumber, points
Set Hole = str.Objects.Get(HoleNumber)
Hole.Main.Attribs.Meshed = False
Hole.Initialize
```

Modeling the supports (spread footings)

The supports are defined by assigning the constraints to the existing structure nodes, in the example below we can see the definition of the support blocked in all six directions, then the support is assigned to the nodes 1 to 4, which correspond to the spread footings.

```
FootName = "Foot"

Set Label = labels.Create(I_LT_SUPPORT, FootName)
Dim footData As RobotNodeSupportData
Set footData = Label.Data
footData.UX = 1
footData.UY = 1
footData.UZ = 1
footData.RX = 1
footData.RY = 1
footData.RZ = 1
labels.Store Label

Dim selectionNodes As RobotSelection
Set selectionNodes = str.Selections.Get(I OT NODE)
selectionNodes.FromText "1 2 3 4"
str.Nodes.SetLabel selectionNodes, I_LT_SUPPORT, FootName
```

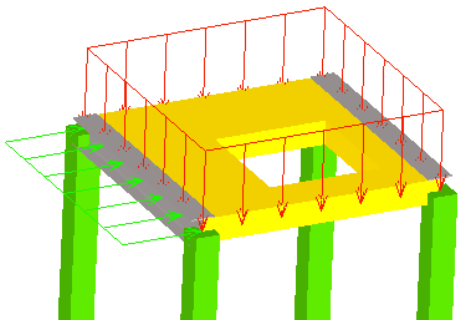
Modeling the ground

If we want to create the ground model with spread footings, we should release the stiffening in *UZ* direction and define the ground elasticity *Kz* in this place. This factor depends on the spread footing geometry and the ground properties and should be calculated by the User, because neither the spread footing geometry nor ground properties belong to the structure model and the program is not able to calculate them. The modeled structure will settle according to the applied parameters.

```
footData.UX = 1
footData.UY = 1
footData.UZ = 0
footData.KZ = 80000000#
footData.RX = 1
footData.RY = 1
footData.RZ = 1
```

Modeling the loads

Definition of the loads applied to the structure starts with defining the load cases of appropriate natures, and then in each case the optional number of loads may be defined. The loads may be applied to the nodes, members, slabs, or may be defined geometrically, in this case the program automatically recognizes the elements to which the load should be applied. In the example three types of loads were defined: self-weight of the whole structure, contour load of live nature, applied to the slab as well as wind load, horizontal, applied to the beam (member No. 5). Self-weight load is not presented in the drawing; it is calculated automatically by the program, basing on structure geometry and material characteristics of the particular elements.



Each load case has its unique number, in the example below self-weight load - 1, live load – 2, wind load – 3.

Defining the self-weight load applied to the whole structure:

```
Dim caseSW As RobotSimpleCase
Set caseSW = str.Cases.CreateSimple(1, "SW", I_CN_PERMANENT,
I_CAT_STATIC_LINEAR)
caseSW.Records.New I_LRT_DEAD
Dim LoadRec As RobotLoadRecord
Set LoadRec = caseSW.Records.Get(1)
LoadRec.SetValue I_DRV_Z, -1
LoadRec.SetValue I_DRV_ENTIRE_STRUCTURE, True
```

Defining the uniform load on contour, applied to the slab. The load is added to „Live” case, which is created for that purpose. The load is in the opposite direction to *Z* axis direction and its value is 10 [kN].

```
Dim LoadRecord As RobotLoadRecord
```

```
Dim CaseLive As RobotSimpleCase
Set CaseLive = str.Cases.CreateSimple(2, "Live", I_CN_EXPLOATATION,
I_CAT_STATIC_LINEAR)
Uniform = CaseLive.Records.New(I_LRT_UNIFORM)
Set LoadRecord = CaseLive.Records.Get(Uniform)
LoadRecord.SetValue I URV PX, 0
LoadRecord.SetValue I URV PY, 0
LoadRecord.SetValue I_URV_PZ, -10000
LoadRecord.Objects.FromText (ObjNumber)
```

Defining the uniform linear load, applied to the member. The load is added to „Wind" live case, which is created for that purpose. The load is in the same direction as *Y* axis direction and its value is 1 [kN].

```
Dim CaseWind As RobotSimpleCase
Set CaseWind = str.Cases.CreateSimple(3, "Wind", I CN WIND,
I CAT STATIC LINEAR)
Uniform = CaseWind.Records.New(I_LRT_BAR_UNIFORM)
Set LoadRecord = CaseWind.Records.Get(Uniform)
LoadRecord.SetValue I BURV PX, 0
LoadRecord.SetValue I_BURV_PY, 1000
LoadRecord.SetValue I_BURV_PZ, 0
LoadRecord.Objects.FromText ("5")
```

Defining the regulations for load combinations

ROS automatically calculates the results for load combinations, according to the applied regulations appropriate for the given national standard. The regulations may be selected by means of project preferences. It is also possible to modify the relations between the load cases, e.g. forcing 2 live cases to be treated as occurring always simultaneously, however in most structures it is possible to work with automatic settings.

```
ProjectPrefs.SetActiveCode(I CT CODE COMBINATIONS, "BAEL 93");
```

LAUNCHING THE CALCULATIONS

Having defined the structure and the loads we may run the calculations. In the simplest case it means calling *Calculate* command.

```
Kernel.CalcEngine.Calculate
```

Since the calculations are complicated and long-lasting process, it is reasonable to take advantage of the advanced options of *ROS* component, such as displaying the progress bar window of the calculations and reporting the errors and warnings.

Error messages are sent from *CalcEngine* component in the form of standard events consistent with *COM* standard. These messages may be ignored or displayed in the form of descriptions comprehensible to the User, what considerably facilitates diagnosing the problems.

The progress bar makes the User aware of the current progress of calculations and estimated time left as wells as enables to abort calculations at the User's request. The progress bar may not appear in spite of activating this option – if the structure is not big and calculation time is negligible. In this case *ROS* will decide about displaying or not displaying the status bar window. In the example described in this manual the status bar window doesn't appear, in order to make it appear we have to define more elements in the structure or reduce the size of finite element, e.g. to 5 [cm], see Parameters of mesh generation chapter.

Structural analysis

Starting the static calculations of the structure with error reports and progress bar displayed.

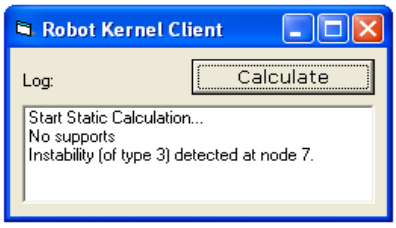
```
Private WithEvents CalcEngine As RobotCalcEngine
-------------------------------------------------------------------------
----
Private Sub CalcEngine_CalcNotifyEx(ByVal nCaller As Long, ByVal
strText As String, ByVal strFullText As String, ByVal strCaption As
String, ByVal nType As Long, ByVal nDataType As Long, ByVal
strSelection As String, bHandled As Boolean, nReturnValue As Long)
If strText <> Empty Then
    bHandled = True
    Output.AddItem strText
End If
-------------------------------------------------------------------------
----
End Sub
Output.AddItem "Start Static Calculation..."
Kernel.CalcEngine.GenerationParams.GenerateNodes BarsAndFiniteElems =
True
Kernel.CalcEngine.GenerateModel
Set CalcEngine = Kernel.CalcEngine
CalcEngine.UseStatusWindow = True
CalcEngine.StatusWindowParent = Form1.hWnd
If CalcEngine.Calculate = False Then
    Output.AddItem "Failed!"
Else
    Output.AddItem "Done!"
End If
Set CalcEngine = Nothing
```

CalcEngine_CalcNotifyEx function is generated automatically in Visual Basic environment. The application has only to display the received message in the form of dialog window or add it to report dialog window. The application may also abort the calculations as an error occurs, when *nReturnValue* is set in appropriate way, e.g.:

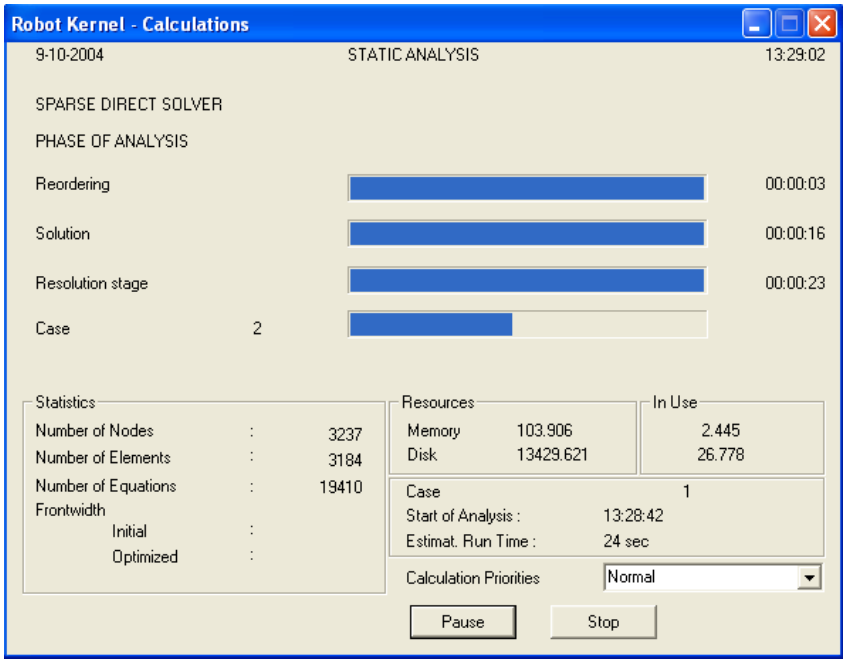
```
Private Sub CalcEngine_CalcNotifyEx( . . . )
```

```
If strText <> Empty Then
    bHandled = True
    nReturnValue = MsgBox(strFullText, nType, "Error")
End If
End Sub
```

The drawing below presents the dialog window of the program that has run calculations of the structure with no supports defined. In this case calculations end with error message.



If after preliminary verification of the structure *ROS* starts the calculations, in case of large structures the progress bar window may appear, as shown in the drawing below.



Calculating the slab reinforcement

Calculating the theoretical areas of reinforcement for concrete slabs is the additional functionality of *ROS* component. To perform these calculations first we should always complete the static calculations of the structure. The calculations are performed run for each slab separately or together for any number of slabs, if their calculating options are identical.

```
Dim concrCalcEngine As RConcrCalcEngine
Set concrCalcEngine = Kernel.ConcrReinfEngine

Dim concrSlabRequiredReinfEngine As RConcrSlabRequiredReinfEngine
Set concrSlabRequiredReinfEngine = concrCalcEngine.SlabRequiredReinf
```

Defining the calculation options for slab reinforcement and assigning them to the slab.

```
Dim slabRnfParams As RConcrSlabRequiredReinfCalcParams
Set slabRnfParams = concrSlabRequiredReinfEngine.Params

slabRnfParams.Method = I_RCM_WOOD_ARMER
slabRnfParams.GloballyAvgDesginForces = False
slabRnfParams.ForcesReduction = False
slabRnfParams.DisplayErrors = False

Dim slabs As RobotSelection
Set slabs = slabRnfParams.Panels

slabs.FromText ObjNumber

SlabReinforcementName = "Slab X"
Set Label = labels.Create(I_LT_PANEL_REINFORCEMENT,
SlabReinforcementName)
Dim rnfData As RConcrReinforceData
labels.Store Label
slab.SetLabel I_LT_PANEL_REINFORCEMENT, SlabReinforcementName
slab.Update
```

Running the calculations of theoretical areas of reinforcement for the selected slab.

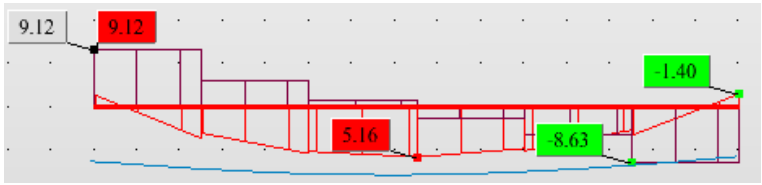
```
If concrSlabRequiredReinfEngine.Calculate = False Then
    Output.AddItem "Failed!"
Else
    Output.AddItem "Done!"
End If
```

GETTING THE RESULTS

ROS makes available a wide range of results for nodes, members, finite elements, seismic calculations, etc. for all load cases and their combinations. Large amount and diversity of results requires the advanced methods of analysis, storage and displaying, in the form, which is convenient and clear to the User. In the example we get only some characteristic values and display them in the form of short messages.

Getting the results for members

In case of the members, the most important are the values of lateral forces, bending moments and deflections. In the example we display the span moment, deflection and reactions of the supports for one of the beams as well as set of forces in the head of one of the columns.
In case of the members, the results are available in the form of diagrams of the internal forces. We may get any value at any point along the member length. In our example the structure is symmetrical so the maximum values for the beams, with vertical load, are in the middle of the structure height. Therefore we get *My* (bending moment) and *Uz* (deflection in vertical direction) values.




```
Text = "  My = " & str.Results.Bars.Forces.Value(5, 2, 0.5).MY / 1000 &
_
     " [kN*m], Qz = " & -str.Results.Bars.Deflections.Value(5, 2,
0.5).UZ _
       * 1000 & " [mm]"
Output.AddItem Text
Text = "  Fz1 = " & str.Results.Bars.Forces.Value(5, 2, 0#).FZ / 1000 &
_
       "  Fz2 = " & str.Results.Bars.Forces.Value(5, 2, 1#).FZ / 1000 &
" [kN]"
```

The obtained values:
My = 5,16 [kN*m], Qz = 0,0788 [mm]
Fz1 = 9,12, Fz2 = -8,63 [kN]
In case of the columns, in order to obtain the results in their heads, we should know whether the element was defined from bottom to top, or from top to bottom and then we should get the results from appropriate end of the bar. In our case the columns were defined from bottom to top, so we should set 1m, what means the column's head, as the relative position, in which we want to obtain the results.

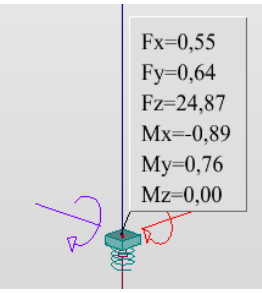
```
Text = "  Fx = " & str.Results.Bars.Forces.Value(4, 3, 1#).FX / 1000 &

   "  Fy = " & str.Results.Bars.Forces.Value(4, 3, 1#).FY / 1000 & "
[kN]"
```

The obtained values:
Fx = 0,89 Fy = -0,75 [kN]

Getting the results for nodes

We will obtain the reactions on the spread footings getting the values from the corresponding nodes of the structure, e.g. :

```
Text = "  Fx = " & str.Results.Nodes.Reactions.Value(1, 1).FX / 1000 &
_
       "  Fy = " & str.Results.Nodes.Reactions.Value(1, 1).FY / 1000 &
_
       "  Fz = " & str.Results.Nodes.Reactions.Value(1, 1).FZ / 1000 &
_
       " [kN]" & _
       "  Mx = " & str.Results.Nodes.Reactions.Value(1, 1).MX / 1000 &

       "  My = " & str.Results.Nodes.Reactions.Value(1, 1).MY / 1000 &

       "  Mz = " & str.Results.Nodes.Reactions.Value(1, 1).MZ / 1000 &

       " [kN*m]"
```



The obtained values:
Fx = 0,548 Fy = 0,643 Fz = 24,87 [kN] Mx = -0,89 My = 0,76 Mz = 2,22-03 [kN*m]

Getting the results for slab reinforcement

In the example we will get the list of nodes of the slab finite elements and values of theoretical areas of reinforcement as well as we will display the maximum values in the slab.

```
Dim SelectionFE As RobotSelection
Set SelectionFE = str.Selections.Get(I OT FINITE ELEMENT)
SelectionFE.FromText (slab.FiniteElems)
Dim ObjFEs As RobotLabelCollection
Set ObjFEs = str.FiniteElems.GetMany(SelectionFE)
Dim AxP As Double, AxM As Double, AyP As Double, AyM As Double
AxP = 0
AyP = 0
AxM = 0
AyM = 0
For N = 1 To ObjFEs.Count
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AX_BOTTOM
    If A > AxM Then
        AxM = A
    End If
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AX_TOP
    If A > AxP Then
```

```
        AxP = A
    End If
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AY_BOTTOM
    If A > AyM Then
        AyM = A
    End If
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AY_TOP
    If A > AyP Then
        AyP = A
    End If
Next N

Output.AddItem "Slab 1, Reinforcemet extreme values:"
Text = "  Ax+ = " & AxP * 10000 & ", Ax- = " & AxM * 10000 & _
       "  Ay+ = " & AyP * 10000 & ", Ay- = " & AyM * 10000 & " [cm2]"
Output.AddItem Text
```

The obtained values:
Ax+ = 1,584, Ax- = 1,584 Ay+ = 1,512, Ay- = 1,512 [cm2]

THE PROGRAM CODE

This chapter includes the complete code of the program discussed in the manual. In spite of Visual Basic code, the next chapter includes the analogous program in C++ language.

Visual Basic

```
Private WithEvents CalcEngine As RobotCalcEngine
'-------------------------------------------------------------------
---
Private Sub CalcEngine_CalcNotifyEx(ByVal nCaller As Long, ByVal
strText As String, ByVal strFullText As String, ByVal strCaption As
String, ByVal nType As Long, ByVal nDataType As Long, ByVal
strSelection As String, bHandled As Boolean, nReturnValue As Long)
If strText <> Empty Then
    Output.AddItem strText
    bHandled = True
End If
End Sub
'-------------------------------------------------------------------
---
Private Sub Go_Click()

Output.Clear
Output.AddItem "Launching Robot Millennium..."

Dim Robot As New RobotApplication
Robot.Project.New I_PT_SHELL

Dim ProjectPrefs As RobotProjectPreferences
Set ProjectPrefs = Robot.Project.Preferences
ProjectPrefs.SetActiveCode I_CT_RC_THEORETICAL_REINF, "BAEL 91"

Dim MeshParams As RobotMeshParams
Set MeshParams = ProjectPrefs.MeshParams
MeshParams.SurfaceParams.Method.Method = I_MMT_DELAUNAY
MeshParams.SurfaceParams.Generation.Type = I_MGT_ELEMENT_SIZE
MeshParams.SurfaceParams.Generation.ElementSize = 0.5
MeshParams.SurfaceParams.Delaunay.Type = I MDT DELAUNAY

Output.AddItem "Structure Generation..."
Dim str As IRobotStructure
Set str = Robot.Project.Structure

With str.Nodes
    .Create 1, 0, 0, 0
    .Create 2, 3, 0, 0
    .Create 3, 3, 3, 0
    .Create 4, 0, 3, 0
    .Create 5, 0, 0, 4
    .Create 6, 3, 0, 4
    .Create 7, 3, 3, 4
    .Create 8, 0, 3, 4
End With
```

```
With str.Bars
    .Create 1, 1, 5
    .Create 2, 2, 6
    .Create 3, 3, 7
    .Create 4, 4, 8
    .Create 5, 5, 6
    .Create 6, 7, 8
End With

Dim labels As RobotLabelServer
Set labels = str.labels

ColumnSectionName = "Rect. Column 30*30"

Set Label = labels.Create(I_LT_BAR_SECTION, ColumnSectionName)
Dim section As RobotBarSectionData
Set section = Label.Data
section.ShapeType = I_BSST_CONCR_COL_R
Dim concrete As RobotBarSectionConcreteData
Set concrete = section.concrete
concrete.SetValue I BSCDV COL B, 0.3
concrete.SetValue I_BSCDV_COL_H, 0.3
section.CalcNonstdGeometry
labels.Store Label

Dim selectionBars As RobotSelection
Set selectionBars = str.Selections.Get(I_OT_BAR)

selectionBars.FromText ("1 2 3 4")
str.Bars.SetLabel selectionBars, I_LT_BAR_SECTION, ColumnSectionName

Dim steelSections As RobotSectionDatabaseList
Set steelSections = ProjectPrefs.SectionsActive
If steelSections.Add("RCAT") = False Then
    Output.AddItem "Steel section base RCAT not found..."
End If

selectionBars.FromText ("5 6")
Set Label = str.labels.Create(I_LT_BAR_SECTION, "HEA 340")
str.labels.Store Label
str.Bars.SetLabel selectionBars, I_LT_BAR_SECTION, "HEA 340"

MaterialName = "Concrete 30"

Set Label = labels.Create(I_LT_MATERIAL, MaterialName)

Dim Material As RobotMaterialData
Set Material = Label.Data
Material.Type = I_MT_CONCRETE
Material.E = 30000000000# ' Young
Material.NU = 1# / 6# ' Poisson
Material.RO = 25000# ' Unit weight
Material.Kirchoff = Material.E / (2 * (1 + Material.NU))
labels.Store Label

Dim points As RobotPointsArray
```

```
Set points = Robot.CmpntFactory.Create(I CT POINTS ARRAY)
points.SetSize 5
With points
    .Set 1, 0, 0, 4
    .Set 2, 3, 0, 4
    .Set 3, 3, 3, 4
    .Set 4, 0, 3, 4
    .Set 5, 0, 0, 4
End With

SlabSectionName = "Slab 30"
Set Label = labels.Create(I_LT_PANEL_THICKNESS, SlabSectionName)

Dim thickness As RobotThicknessData
Set thickness = Label.Data
thickness.MaterialName = MaterialName
thickness.ThicknessType = I_TT_HOMOGENEOUS
Dim thicknessData As RobotThicknessHomoData
Set thicknessData = thickness.Data
thicknessData.ThickConst = 0.3
labels.Store Label

Dim slab As RobotObjObject
ObjNumber = str.Objects.FreeNumber
str.Objects.CreateContour ObjNumber, points
Set slab = str.Objects.Get(ObjNumber)
slab.Main.Attribs.Meshed = True
slab.SetLabel I_LT_PANEL_THICKNESS, SlabSectionName
slab.Initialize

With points
    .Set 1, 1.1, 1.1, 4
    .Set 2, 2.5, 1.1, 4
    .Set 3, 2.5, 2.5, 4
    .Set 4, 1.1, 2.5, 4
    .Set 5, 1.1, 1.1, 4
End With

Dim Hole As RobotObjObject
HoleNumber = str.Objects.FreeNumber
str.Objects.CreateContour HoleNumber, points
Set Hole = str.Objects.Get(HoleNumber)
Hole.Main.Attribs.Meshed = False
Hole.Initialize

FootName = "Foot"

Set Label = labels.Create(I_LT_SUPPORT, FootName)
Dim footData As RobotNodeSupportData
Set footData = Label.Data
footData.UX = 1
footData.UY = 1
footData.UZ = 0
footData.KZ = 80000000#
footData.RX = 1
footData.RY = 1
footData.RZ = 1
```

```
labels.Store Label

Dim SelectionNodes As RobotSelection
Set SelectionNodes = str.Selections.Get(I_OT_NODE)
SelectionNodes.FromText "1 2 3 4"
str.Nodes.SetLabel SelectionNodes, I LT SUPPORT, FootName

'self weight on entire structure
Dim caseSW As RobotSimpleCase
Set caseSW = str.Cases.CreateSimple(1, "SW", I CN PERMANENT,
I CAT STATIC LINEAR)
caseSW.Records.New I_LRT_DEAD
Dim LoadRec As RobotLoadRecord
Set LoadRec = caseSW.Records.Get(1)
LoadRec.SetValue I_DRV_Z, -1
LoadRec.SetValue I_DRV_ENTIRE_STRUCTURE, True

'contour live load on the slab
Dim LoadRecord As RobotLoadRecord
Dim CaseLive As RobotSimpleCase
Set CaseLive = str.Cases.CreateSimple(2, "Live", I_CN_EXPLOATATION,
I CAT STATIC LINEAR)
Uniform = CaseLive.Records.New(I_LRT_UNIFORM)
Set LoadRecord = CaseLive.Records.Get(Uniform)
LoadRecord.SetValue I_URV_PX, 0
LoadRecord.SetValue I URV PY, 0
LoadRecord.SetValue I_URV_PZ, -10000
'apply created load to the slab
LoadRecord.Objects.FromText (ObjNumber)

'linear wind load on the beam
Dim CaseWind As RobotSimpleCase
Set CaseWind = str.Cases.CreateSimple(3, "Wind", I_CN_WIND,
I_CAT_STATIC_LINEAR)
Uniform = CaseWind.Records.New(I LRT BAR UNIFORM)
Set LoadRecord = CaseWind.Records.Get(Uniform)
LoadRecord.SetValue I BURV PX, 0
LoadRecord.SetValue I_BURV_PY, 1000
LoadRecord.SetValue I BURV PZ, 0
'apply created load to the beam
LoadRecord.Objects.FromText ("5")

Output.AddItem "Start Static Calculation..."
Set CalcEngine = Robot.Project.CalcEngine
CalcEngine.GenerationParams.GenerateNodes BarsAndFiniteElems = True
CalcEngine.UseStatusWindow = True
CalcEngine.StatusWindowParent = Form1.hWnd
If CalcEngine.Calculate = False Then
    Output.AddItem "Failed!"
Else
    Output.AddItem "Done!"
End If
Set CalcEngine = Nothing

Dim concrCalcEngine As RConcrCalcEngine
Set concrCalcEngine = Robot.Project.ConcrReinfEngine
```

```
Dim concrSlabRequiredReinfEngine As RConcrSlabRequiredReinfEngine
Set concrSlabRequiredReinfEngine = concrCalcEngine.SlabRequiredReinf

Dim slabRnfParams As RConcrSlabRequiredReinfCalcParams
Set slabRnfParams = concrSlabRequiredReinfEngine.Params

slabRnfParams.Method = I RCM WOOD ARMER
slabRnfParams.GloballyAvgDesginForces = False
slabRnfParams.ForcesReduction = False
slabRnfParams.DisplayErrors = False
slabRnfParams.CasesULS.FromText ("1 2 3 4 5 6 7 8")

Dim slabs As RobotSelection
Set slabs = slabRnfParams.Panels

slabs.FromText ObjNumber

SlabReinforcementName = "Slab X"
Set Label = labels.Create(I_LT_PANEL_REINFORCEMENT,
SlabReinforcementName)
labels.Store Label
slab.SetLabel I LT PANEL REINFORCEMENT, SlabReinforcementName
slab.Update

Output.AddItem "Start Slab Reinforcement Calculation..."
If concrSlabRequiredReinfEngine.Calculate = False Then
    Output.AddItem "Failed!"
Else
    Output.AddItem "Done!"
End If

'getting results My and Yz for beam (bar 5) with live load (case 2)
Dim Text As String
Output.AddItem "Bar 5, Live:"
Text = "  My = " & str.Results.Bars.Forces.Value(5, 2, 0.5).MY / 1000 &
_
       " [kN*m], Qz = " & -str.Results.Bars.Deflections.Value(5, 2,
0.5).UZ _
       * 1000 & " [mm]"
Output.AddItem Text
Text = "  Fz1 = " & str.Results.Bars.Forces.Value(5, 2, 0#).FZ / 1000 &
_
       "  Fz2 = " & str.Results.Bars.Forces.Value(5, 2, 1#).FZ / 1000 &
" [kN]"
Output.AddItem Text

'getting results Fx and Fy for column (bar 4) with wind load (case 3)
Output.AddItem "Bar 4, Wind:"
Text = "  Fx = " & str.Results.Bars.Forces.Value(4, 3, 1#).FX / 1000 &
_
       "  Fy = " & str.Results.Bars.Forces.Value(4, 3, 1#).FY / 1000 &

       " [kN]"
Output.AddItem Text

'getting results Fx, Fy, Fz, Mx, My, Mz for foot (node 1) with self-
weight (case 1)
```

```
Output.AddItem "Node 1, Self-Weight:"
Text = "  Fx = " & str.Results.Nodes.Reactions.Value(1, 1).FX / 1000 &
_
       "  Fy = " & str.Results.Nodes.Reactions.Value(1, 1).FY / 1000 &

       "  Fz = " & str.Results.Nodes.Reactions.Value(1, 1).FZ / 1000 &

       " [kN]" & _
       "  Mx = " & str.Results.Nodes.Reactions.Value(1, 1).MX / 1000 &

       "  My = " & str.Results.Nodes.Reactions.Value(1, 1).MY / 1000 &
_
       "  Mz = " & str.Results.Nodes.Reactions.Value(1, 1).MZ / 1000 &

       " [kN*m]"
Output.AddItem Text

'getting results Ax+, Ax-, Ay+, Ay- for slab
Dim SelectionFE As RobotSelection
Set SelectionFE = str.Selections.Get(I OT FINITE ELEMENT)
SelectionFE.FromText (slab.FiniteElems)
Dim ObjFEs As RobotLabelCollection
Set ObjFEs = str.FiniteElems.GetMany(SelectionFE)
Dim AxP As Double, AxM As Double, AyP As Double, AyM As Double
AxP = 0
AyP = 0
AxM = 0
AyM = 0
For N = 1 To ObjFEs.Count
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AX_BOTTOM
    If A > AxM Then
        AxM = A
    End If
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AX_TOP
    If A > AxP Then
        AxP = A
    End If
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AY_BOTTOM
    If A > AyM Then
        AyM = A
    End If
    A =
str.Results.FiniteElems.Reinforcement(ObjFEs.Get(N).Number).AY_TOP
    If A > AyP Then
        AyP = A
    End If
Next N

Output.AddItem "Slab 1, Reinforcemet extreme values:"
Text = "  Ax+ = " & AxP * 10000 & ", Ax- = " & AxM * 10000 & _
       "  Ay+ = " & AyP * 10000 & ", Ay- = " & AyM * 10000 & " [cm2]"
Output.AddItem Text

Set Robot = Nothing
```

```
Output.AddItem "... Robot Closed"

End Sub
```

C++

The example was written taking advantage of Microsoft Visual C++ 6.0 compiler and MFC 4.2 library.

```
#include "stdafx.h"
#include "KernelForDummiesC.h"
#include <afxctl.h>

#ifdef _DEBUG
#define new DEBUG NEW
#endif

#import <RobotOM.tlb> no_namespace
using namespace std;

class CRobotCalcEngineEvents : public CCmdTarget
{
   DECLARE_DYNAMIC(CRobotCalcEngineEvents)
   CRobotCalcEngineEvents(IRobotCalcEngine* pRobotCalcEngine);
   virtual ~CRobotCalcEngineEvents();

// Operations
public:
   void OnCalcNotifyEx(long nCaller, LPCTSTR lpszText, LPCTSTR
lpszFullText, LPCTSTR lpszCaption, long nType, long nDataType, LPCTSTR
lpszSelection, BOOL* bHandled, long* nReturnValue);

// Implementation
protected:
   DWORD m_dwCookie;
   IRobotCalcEngine* m_pRobotCalcEngine;
   DECLARE_DISPATCH_MAP()
   DECLARE_INTERFACE_MAP()
};

// The one and only application object

CWinApp theApp;

using namespace std;

int  tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
   // initialize MFC and print and error on failure
   if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(),
0))
   {
       tprintf( T("Fatal Error: MFC initialization failed\n"));
      return 1;
   }

   ::CoInitialize(NULL);
```

```
   cout << "Launching Robot Millennium..." << endl;
   IRobotApplicationPtr pRobot(__uuidof(RobotApplication));
   pRobot->GetProject()->New(I_PT_SHELL);

   IRobotProjectPreferencesPtr pProjectPrefs = pRobot->GetProject()-
>GetPreferences();
   pProjectPrefs->SetActiveCode(I CT RC THEORETICAL REINF, "BAEL 91");

   IRobotMeshParamsPtr pMeshParams = pProjectPrefs->GetMeshParams();
   pMeshParams->GetSurfaceParams()->GetMethod()-
>PutMethod(I MMT DELAUNAY);
   pMeshParams->GetSurfaceParams()->GetGeneration()-
>PutType(I_MGT_ELEMENT_SIZE);
   pMeshParams->GetSurfaceParams()->GetGeneration()-
>PutElementSize(0.5);
   pMeshParams->GetSurfaceParams()->GetDelaunay()-
>PutType(I_MDT_DELAUNAY);

   cout << "Structure Generation..." << endl;
   IRobotStructurePtr pStr = pRobot->GetProject()->GetStructure();

   IRobotNodeServerPtr pNodes = pStr->GetNodes();
   pNodes->Create(1, 0, 0, 0);
   pNodes->Create(2, 3, 0, 0);
   pNodes->Create(3, 3, 3, 0);
   pNodes->Create(4, 0, 3, 0);
   pNodes->Create(5, 0, 0, 4);
   pNodes->Create(6, 3, 0, 4);
   pNodes->Create(7, 3, 3, 4);
   pNodes->Create(8, 0, 3, 4);

   IRobotBarServerPtr pBars = pStr->GetBars();
   pBars->Create(1, 1, 5);
   pBars->Create(2, 2, 6);
   pBars->Create(3, 3, 7);
   pBars->Create(4, 4, 8);
   pBars->Create(5, 5, 6);
   pBars->Create(6, 7, 8);

   IRobotLabelServerPtr pLabels = pStr->GetLabels();

   string ColumnSectionName = "Rect. Column 30*30";

   IRobotLabelPtr pLabel = pLabels->Create(I_LT_BAR_SECTION,
ColumnSectionName.c str());
   IRobotBarSectionDataPtr pSection = pLabel->GetData();
   pSection->PutShapeType(I_BSST_CONCR_COL_R);
   IRobotBarSectionConcreteDataPtr pConcrete = pSection->GetConcrete();
   pConcrete->SetValue(I BSCDV COL B, 0.3);
   pConcrete->SetValue(I_BSCDV_COL_H, 0.3);
   pSection->CalcNonstdGeometry();
   pLabels->Store(pLabel);

   IRobotSelectionPtr pSelectionBars = pStr->GetSelections()-
>Get(I_OT_BAR);
   pSelectionBars->FromText("1 2 3 4");
```

```
   pBars->SetLabel(pSelectionBars, I LT BAR SECTION,
ColumnSectionName.c_str());

   IRobotSectionDatabaseListPtr pSteelSections = pProjectPrefs-
>GetSectionsActive();
   if (pSteelSections->Add("RCAT") != VARIANT TRUE)
      cout << "Steel section base RCAT not found..." << endl;

   pSelectionBars->FromText ("5 6");
   pLabel = pLabels->Create(I LT BAR SECTION, "HEA 340");
   pLabels->Store(pLabel);
   pBars->SetLabel(pSelectionBars, I_LT_BAR_SECTION, "HEA 340");

   string MaterialName = "Concrete 30";

   pLabel = pLabels->Create(I_LT_MATERIAL, MaterialName.c_str());

   IRobotMaterialDataPtr pMaterial = pLabel->GetData();
   pMaterial->PutType(I_MT_CONCRETE);
   pMaterial->PutE(30000000000); // Young
   pMaterial->PutNU(1. / 6.); // Poisson
   pMaterial->PutRO(25000.); // Unit weight
   pMaterial->PutKirchoff(pMaterial->GetE() / (2 * (1 + pMaterial-
>GetNU())));
   pLabels->Store(pLabel);

   IRobotPointsArrayPtr pPoints = pRobot->GetCmpntFactory()-
>Create(I_CT_POINTS_ARRAY);
   pPoints->SetSize(5);
   pPoints->Set(1, 0, 0, 4);
   pPoints->Set(2, 3, 0, 4);
   pPoints->Set(3, 3, 3, 4);
   pPoints->Set(4, 0, 3, 4);
   pPoints->Set(5, 0, 0, 4);

   string SlabSectionName = "Slab 30";
   pLabel = pLabels->Create(I LT PANEL THICKNESS,
SlabSectionName.c_str());

   IRobotThicknessDataPtr pThickness = pLabel->GetData();
   pThickness->PutMaterialName(MaterialName.c_str());
   pThickness->PutThicknessType(I_TT_HOMOGENEOUS);
   IRobotThicknessHomoDataPtr pThicknessData = pThickness->GetData();
   pThicknessData->PutThickConst(0.3);
   pLabels->Store(pLabel);

   long ObjNumber = pStr->GetObjects()->GetFreeNumber();
   pStr->GetObjects()->CreateContour(ObjNumber, pPoints);
   IRobotObjObjectPtr pSlab = pStr->GetObjects()->Get(ObjNumber);
   pSlab->GetMain()->GetAttribs()->PutMeshed(VARIANT_TRUE);
   pSlab->SetLabel(I_LT_PANEL_THICKNESS, SlabSectionName.c_str());
   pSlab->Initialize();

   pPoints->Set(1, 1.1, 1.1, 4);
   pPoints->Set(2, 2.5, 1.1, 4);
   pPoints->Set(3, 2.5, 2.5, 4);
   pPoints->Set(4, 1.1, 2.5, 4);
```

```
   pPoints->Set(5, 1.1, 1.1, 4);

   long HoleNumber = pStr->GetObjects()->GetFreeNumber();
   pStr->GetObjects()->CreateContour(HoleNumber, pPoints);
   IRobotObjObjectPtr pHole = pStr->GetObjects()->Get(HoleNumber);
   pHole->GetMain()->GetAttribs()->PutMeshed(VARIANT FALSE);
   pHole->Initialize();

   string FootName = "Foot";

   pLabel = pLabels->Create(I LT SUPPORT, FootName.c str());
   IRobotNodeSupportDataPtr pFootData = pLabel->GetData();
   pFootData->PutUX(1.);
   pFootData->PutUY(1.);
   pFootData->PutUZ(0.);
   pFootData->PutKZ(80000000.);
   pFootData->PutRX(1.);
   pFootData->PutRY(1.);
   pFootData->PutRZ(1.);
   pLabels->Store(pLabel);

   IRobotSelectionPtr pSelectionNodes = pStr->GetSelections()-
>Get(I_OT_NODE);
   pSelectionNodes->FromText("1 2 3 4");
   pStr->GetNodes()->SetLabel(pSelectionNodes, I_LT_SUPPORT,
FootName.c str());

   //self weight on entire structure
   IRobotSimpleCasePtr pCaseSW = pStr->GetCases()->CreateSimple(1,
"SW", I_CN_PERMANENT, I_CAT_STATIC_LINEAR);
   pCaseSW->GetRecords()->New(I_LRT_DEAD);
   IRobotLoadRecordPtr pLoadRecord = pCaseSW->GetRecords()->Get(1);
   pLoadRecord->SetValue(I_DRV_Z, -1);
   pLoadRecord->SetValue(I_DRV_ENTIRE_STRUCTURE, VARIANT_TRUE);

   //contour live load on the slab
   IRobotSimpleCasePtr pCaseLive = pStr->GetCases()->CreateSimple(2,
"Live", I_CN_EXPLOATATION, I_CAT_STATIC_LINEAR);
   long Uniform = pCaseLive->GetRecords()->New(I LRT UNIFORM);
   pLoadRecord = pCaseLive->GetRecords()->Get(Uniform);
   pLoadRecord->SetValue(I_URV_PX, 0.);
   pLoadRecord->SetValue(I_URV_PY, 0.);
   pLoadRecord->SetValue(I_URV_PZ, -10000.);
   //apply created load to the slab
   CString buff;
   buff.Format("%d", ObjNumber);
   pLoadRecord->GetObjects()->FromText((LPCSTR)buff);

   //linear wind load on the beam
   IRobotSimpleCasePtr pCaseWind = pStr->GetCases()->CreateSimple(3,
"Wind", I_CN_WIND, I_CAT_STATIC_LINEAR);
   Uniform = pCaseWind->GetRecords()->New(I LRT BAR UNIFORM);
   pLoadRecord = pCaseWind->GetRecords()->Get(Uniform);
   pLoadRecord->SetValue(I BURV PX, 0.);
   pLoadRecord->SetValue(I_BURV_PY, 1000.);
   pLoadRecord->SetValue(I_BURV_PZ, 0.);
   //apply created load to the beam
```

```
   pLoadRecord->GetObjects()->FromText("5");

   cout << "Start Static Calculation..." << endl;
   IRobotCalcEnginePtr pCalcEngine = pRobot->GetProject()-
>GetCalcEngine();
   pCalcEngine->GetGenerationParams()-
>PutGenerateNodes BarsAndFiniteElems(VARIANT TRUE);
   {
      CRobotCalcEngineEvents evHandler(pCalcEngine);
      pCalcEngine->PutUseStatusWindow(VARIANT TRUE);
      if (pCalcEngine->Calculate() != VARIANT TRUE)
         cout << "Failed!" << endl;
      else
         cout << "Done!" << endl;
   }

   IRConcrCalcEnginePtr pConcrCalcEngine = pRobot->GetProject()-
>GetConcrReinfEngine();

   IRConcrSlabRequiredReinfEnginePtr pConcrSlabRequiredReinfEngine =
pConcrCalcEngine->GetSlabRequiredReinf();

   IRConcrSlabRequiredReinfCalcParamsPtr pSlabRnfParams =
pConcrSlabRequiredReinfEngine->GetParams();

   pSlabRnfParams->PutMethod(I RCM WOOD ARMER);
   pSlabRnfParams->PutGloballyAvgDesginForces(VARIANT_FALSE);
   pSlabRnfParams->PutForcesReduction(VARIANT_FALSE);
   pSlabRnfParams->PutDisplayErrors(VARIANT_FALSE);
   pSlabRnfParams->GetCasesULS()->FromText("1 2 3 4 5 6 7 8");

   IRobotSelectionPtr pSlabs = pSlabRnfParams->GetPanels();
   pSlabs->FromText((LPCSTR)buff);

   string SlabReinforcementName = "Slab X";
   pLabel = pLabels->Create(I_LT_PANEL_REINFORCEMENT,
SlabReinforcementName.c str());
   pLabels->Store(pLabel);
   pSlab->SetLabel(I LT PANEL REINFORCEMENT,
SlabReinforcementName.c_str());
   pSlab->Update();

   cout << "Start Slab Reinforcement Calculation..." << endl;
   if (pConcrSlabRequiredReinfEngine->Calculate() != VARIANT_TRUE)
      cout << "Failed!" << endl;
   else
      cout << "Done!" << endl;

   IRobotResultServerPtr pResults = pStr->GetResults();
   //getting results My and Yz for beam (bar 5) with live load (case 2)
   cout << "Bar 5, Live:" << endl;
   cout << "  My = " << pResults->GetBars()->GetForces()->Value(5, 2,
0.5)->GetMY() / 1000 <<
           " [kN*m], Qz = " << -pResults->GetBars()->GetDeflections()-
>Value(5, 2, 0.5)->GetUZ() * 1000 << " [mm]" << endl;
   cout << "  Fz1 = " << pResults->GetBars()->GetForces()->Value(5, 2,
0.)->GetFZ() / 1000 <<
```

```
           "  Fz2 = " << pResults->GetBars()->GetForces()->Value(5, 2,
1.)->GetFZ() / 1000 << " [kN]" << endl;

   //getting results Fx and Fy for column (bar 4) with wind load (case
3)
   cout << "Bar 4, Wind:" << endl;
   cout << "  Fx = " << pResults->GetBars()->GetForces()->Value(4, 3,
1.)->GetFX() / 1000 <<
           "  Fy = " << pResults->GetBars()->GetForces()->Value(4, 3,
1.)->GetFY() / 1000 << " [kN]" << endl;

   //getting results Fx, Fy, Fz, Mx, My, Mz for foot (node 1) with
self-weight (case 1)
   cout << "Node 1, Self-Weight:" << endl;
   cout << "  Fx = " << pResults->GetNodes()->GetReactions()->Value(1,
1)->GetFX() / 1000 <<
           "  Fy = " << pResults->GetNodes()->GetReactions()->Value(1,
1)->GetFY() / 1000 <<
           "  Fz = " << pResults->GetNodes()->GetReactions()->Value(1,
1)->GetFZ() / 1000 << " [kN]" <<
           "  Mx = " << pResults->GetNodes()->GetReactions()->Value(1,
1)->GetMX() / 1000 <<
           "  My = " << pResults->GetNodes()->GetReactions()->Value(1,
1)->GetMY() / 1000 <<
           "  Mz = " << pResults->GetNodes()->GetReactions()->Value(1,
1)->GetMZ() / 1000 << " [kN*m]" << endl;

   //getting results Ax+, Ax-, Ay+, Ay- for slab
   IRobotSelectionPtr pSelectionFE = pStr->GetSelections()-
>Get(I_OT_FINITE_ELEMENT);
   pSelectionFE->FromText(pSlab->GetFiniteElems());
   IRobotCollectionPtr pObjFEs = pStr->GetFiniteElems()-
>GetMany(pSelectionFE);
   double AxP, AxM, AyP, AyM;
   AxP = 0;
   AyP = 0;
   AxM = 0;
   AyM = 0;
   for (long N = 1; N<pObjFEs->GetCount(); N++)
   {
      IRobotFiniteElementPtr pIRobotFiniteElement = pObjFEs->Get(N);
      long Number = pIRobotFiniteElement->GetNumber();
      AxM = max(AxM, pResults->GetFiniteElems()->Reinforcement(Number,
0)->GetAX_BOTTOM());
      AxP = max(AxP, pResults->GetFiniteElems()->Reinforcement(Number,
0)->GetAX_TOP());
      AyM = max(AyM, pResults->GetFiniteElems()->Reinforcement(Number,
0)->GetAY_BOTTOM());
      AyP = max(AyP, pResults->GetFiniteElems()->Reinforcement(Number,
0)->GetAY_TOP());
   }

   cout << "Slab 1, Reinforcemet extreme values:" << endl;
   cout << "  Ax+ = " << AxP * 10000 << ", Ax- = " << AxM * 10000 <<
           "  Ay+ = " << AyP * 10000 << ", Ay- = " << AyM * 10000 << "
[cm2]" << endl;
```

```
   return 0;
}

CRobotCalcEngineEvents::CRobotCalcEngineEvents(IRobotCalcEngine*
pRobotCalcEngine)
{
   EnableAutomation();  // Needed in order to sink events.
   m_pRobotCalcEngine = pRobotCalcEngine;
   m_dwCookie = 0;
   if (m_pRobotCalcEngine)
      VERIFY(AfxConnectionAdvise(m_pRobotCalcEngine,
__uuidof(_IRobotCalcEngineEvents),
                                 GetInterface(&IID_IUnknown), TRUE,
&m_dwCookie));
}

CRobotCalcEngineEvents::~CRobotCalcEngineEvents()
{
   if (m_dwCookie && m_pRobotCalcEngine)
   {
      VERIFY(AfxConnectionUnadvise(m_pRobotCalcEngine,
  uuidof( IRobotCalcEngineEvents),
                                   GetInterface(&IID_IUnknown), TRUE,
m_dwCookie));
      m_dwCookie = 0;
   }
}

void CRobotCalcEngineEvents::OnCalcNotifyEx(long nCaller, LPCTSTR
lpszText, LPCTSTR lpszFullText, LPCTSTR lpszCaption, long nType, long
nDataType, LPCTSTR lpszSelection, BOOL* bHandled, long* nReturnValue)
{
   if (strlen(lpszText))
   {
      cout << lpszText << endl;
      *bHandled = TRUE;
   }
}

IMPLEMENT_DYNAMIC(CRobotCalcEngineEvents, CCmdTarget)

BEGIN_DISPATCH_MAP(CRobotCalcEngineEvents, CCmdTarget)
   DISP_FUNCTION_ID(CRobotCalcEngineEvents, "CalcNotifyEx", 0x2,
                    OnCalcNotifyEx, VT_EMPTY, VTS_I4 VTS_BSTR VTS_BSTR
VTS_BSTR VTS_I4 VTS_I4 VTS_BSTR VTS_PBOOL VTS_PI4)
END_DISPATCH_MAP()

BEGIN_INTERFACE_MAP(CRobotCalcEngineEvents, CCmdTarget)
   INTERFACE_PART(CRobotCalcEngineEvents,
__uuidof(_IRobotCalcEngineEvents), Dispatch)
END_INTERFACE_MAP()
```